How to Use Bitcoin to Play Decentralized Poker

Iddo Bentov
Technion

Ranjit Kumaresan
MIT

GTACS

Tal Moran
IDC

January 8, 2015
Parties $P_1, P_2, \ldots, P_n$ with inputs $x_1, x_2, \ldots, x_n$ send messages to each other, and wish to securely compute $f(x_1, x_2, \ldots, x_n)$.

Example of SFE:

\[
x_i = (sk_i, c) \\
sk = sk_1 \oplus sk_2 \oplus \cdots \oplus sk_n \\
f(x_1, x_2, \ldots, x_n) = \text{decrypt}(sk, c)
\]

Reactive MPC: think of poker cards
Fairness: if any party receives the output, then all honest parties must receive the output.

"Security with abort" is possible

- Secure MPC is possible [Yao86, GMW87, ...]
  - Security: correctness, privacy, independence of inputs, fairness
  - Even with dishonest majority, in the computational setting.

Full security is impossible

- Fair MPC is impossible [Cle86]
  - $r$-round 2-party coin toss protocol is susceptible to $\Omega(1/r)$ bias.
  - $\Rightarrow$ no fair protocol for XOR, barring gradual release [...]

Impossibility of fair MPC
Our results

Outline of this presentation

1. Impose fairness for any SFE, without an honest majority.
2. Secure (reactive) MPC with money inputs and outputs.
   - Example: poker.
Formal model that incorporates coins

Functionality $\mathcal{F}$ versus functionality $\mathcal{F}^*$ with coins

- If party $P_i$ has (say) secret key $sk_0$ and sends it to party $P_j$, then both $P_i$ and $P_j$ will have the string $sk_0$.

- If party $P_i$ has coins$(x)$ and sends $y < x$ coins to party $P_j$, then $P_i$ will have coins$(x - y)$ and $P_j$ will have extra coins$(y)$.

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.
Formal model that incorporates coins

Functionality $\mathcal{F}_{\square}$ versus functionality $\mathcal{F}_{\square}^\ast$ with coins

- If party $P_i$ has (say) secret key $sk_0$ and sends it to party $P_j$, then both $P_i$ and $P_j$ will have the string $sk_0$.
- If party $P_i$ has coins$(x)$ and sends $y < x$ coins to party $P_j$, then $P_i$ will have coins$(x - y)$ and $P_j$ will have extra coins$(y)$.

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.
- Ideally, all the parties deem coins to be valuable assets.
- Sending coins$(x)$ may require a broadcast that reveals at least the amount $x$ (maybe not in ZK cryptocurrency like Zerocash).
- It is possible to define a “secure computation with coins” model directly, or by using (UC) ideal functionalities.
- We provide simulation based proofs (but not in this talk).
Claim-or-Refund for two parties $P_s, P_r$ (implicit in [Max11],[BBSU12])

The $F^*_{\text{CR}}$ Claim-or-Refund ideal functionality

1. The sender $P_s$ deposits (locks) her coins ($q$) while specifying a time bound $\tau$ and a circuit $\phi(\cdot)$.
2. The receiver $P_r$ can claim (gain possession) of the coins ($q$) by publicly revealing a witness $w$ that satisfies $\phi(w) = 1$.
3. If $P_r$ didn’t claim within time $\tau$, coins ($q$) are refunded to $P_s$. 

How to realize $F^*_{\text{CR}}$ via Bitcoin

Old version: using “timelock” transactions.

New version: OP CHECKLOCKTIMEVERIFY (abbrv. CLTV) enables $F^*_{\text{CR}}$ directly, avoiding transaction malleability attacks.
Claim-or-Refund for two parties $P_s, P_r$ (implicit in [Max11],[BBSU12])

The $\mathcal{F}^*_{\text{CR}}$ Claim-or-Refund ideal functionality

1. The sender $P_s$ deposits (locks) her coins($q$) while specifying a time bound $\tau$ and a circuit $\phi(\cdot)$.
2. The receiver $P_r$ can claim (gain possession) of the coins($q$) by publicly revealing a witness $w$ that satisfies $\phi(w) = 1$.
3. If $P_r$ didn’t claim within time $\tau$, coins($q$) are refunded to $P_s$.

How to realize $\mathcal{F}^*_{\text{CR}}$ via Bitcoin

- Old version: using “timelock” transactions.
- New version: OP_CHECKLOCKTIMEVERIFY (abbrv. CLTV) enables $\mathcal{F}^*_{\text{CR}}$ directly, avoiding transaction malleability attacks.
### Fairness with penalties

### Secure cash distribution

### End

**F^*_{CR} via Bitcoin (without CLTV)**

<table>
<thead>
<tr>
<th>High-level description the F^*_{CR} implementation in Bitcoin</th>
</tr>
</thead>
<tbody>
<tr>
<td>- $P_s$ controls $TX_{old}$ that resides on the blockchain.</td>
</tr>
<tr>
<td>- $P_s$ creates a transaction $TX_{new}$ that spends $TX_{old}$ to a Bitcoin script that can be redeemed by $P_s$ and $P_r$, or only by $P_r$ by supplying a witness $w$ that satisfies $\phi(w) = 1$.</td>
</tr>
<tr>
<td>- $P_s$ asks $P_r$ to sign a timelock transaction that refunds $TX_{new}$ to $P_s$ at time $\tau$ (conditioned upon both $P_s$ and $P_r$ signing).</td>
</tr>
<tr>
<td>- After $P_r$ signs the refund, $P_s$ can safely broadcast $TX_{new}$.</td>
</tr>
</tbody>
</table>

1. $P_s$ is safe because $P_r$ only sees $\text{Hash}(TX_{new})$, and therefore cannot broadcast $TX_{new}$ to cause $P_s$ to lose the coins.

2. $P_r$ can safely sign the random-looking data $\text{Hash}(TX_{new})$ because the protocol uses a freshly generated $(sk_R, pk_R)$ pair.
## The structure of Bitcoin transactions

### How standard Bitcoin transactions are chained

- $TX_{old} = \text{earlier } TX \text{ output of coins}(q) \text{ is redeemable by } pk_A$
- $id_{old} = \text{Hash}(TX_{old})$
- $PREPARE_{new} = (id_{old}, q, pk_B, 0)$  \hspace{1cm} \text{0 means no timelock}
- $TX_{new} = (PREPARE_{new}, \text{Sign}_{sk_A}(PREPARE_{new}))$
- $id_{new} = \text{Hash}(TX_{new})$
- Initial minting transaction specifies some $pk_M$ that belongs to a miner, and is created via \textit{proof of work}.  


Realization of $\mathcal{F}_\text{CR}^*$ via Bitcoin (without CLTV)

The $\mathcal{F}_\text{CR}^*$ transaction

- $\text{PREPARE}_{\text{new}} = (id_{\text{old}}, q, (pk_S \land pk_R) \lor (\phi(\cdot) \land pk_R), 0)$
- $\phi(\cdot)$ can be $\text{SHA256}(\cdot) == Y$ where $Y$ is hardcoded.
- $TX_{\text{new}} = (\text{PREPARE}_{\text{new}}, \text{Sign}_{sk_S}(\text{PREPARE}_{\text{new}}))$
- $id_{\text{new}} = \text{Hash}(TX_{\text{new}})$
- $P_s$ sends $\text{PREPARE}_{\text{refund}} = (id_{\text{new}}, q, pk_S, \tau)$ to $P_r$
- $P_r$ sends $\sigma_R = \text{Sign}_{sk_R}(\text{PREPARE}_{\text{refund}})$ to $P_s$
- $P_s$ broadcasts $TX_{\text{new}}$ to the Bitcoin network
- If $P_r$ doesn’t reveal $w$ until time $\tau$ then $P_s$ creates $TX_{\text{refund}} = (\text{PREPARE}_{\text{refund}}, (\text{Sign}_{sk_S}(\text{PREPARE}_{\text{refund}}), \sigma_R))$ and broadcasts it to reclaim her $q$ coins
Secure cash distribution via Bitcoin with CLTV (operational since ≈ December 2015)

Pseudocode: \( pk_S, pk_R, h_0, \tau \) are hardcoded

if (block# > \( \tau \)) then
  \( P_s \) can spend the coins\((q)\) by signing with \( sk_s \)
else
  \( P_r \) can spend the coins\((q)\) by signing with \( sk_r \)
  AND
  supplying \( w \) such that \( \text{Hash}(w) = h_0 \)
  \( \leftarrow \text{this is } \phi(\cdot) \)

Bitcoin script:

\[
\text{IF } \langle \text{timeout} \rangle \text{ CHECKLOCKTIMEVERIFY }
\text{ HASH256 } \langle h_0 \rangle \text{ EQUALVERIFY } \langle pk_r \rangle \text{ CHECKSIGVERIFY }
\text{ELSE }
\langle pk_s \rangle \text{ CHECKSIGVERIFY }
\text{ENDIF}
\]
Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn’t deliver output to honest parties $\Rightarrow$ every honest party is compensated
Fairness with penalties

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn’t deliver output to honest parties $\Rightarrow$ every honest party is compensated

Outline of $\mathcal{F}_f^*$ – fairness with penalties for any function $f$

- $P_1, \ldots, P_n$ with $x_1, \ldots, x_n$ run secure unfair SFE for $f$ that
  1. Computes additive shares $(y_1, \ldots, y_n)$ of $y = f(x_1, \ldots, x_n)$
  2. Computes $Tags = (\text{com}(y_1), \ldots, \text{com}(y_n)) = (\text{hash}(y_1), \ldots, \text{hash}(y_n))$
  3. Delivers $(y_i, Tags)$ to every $P_i$

- $P_1, \ldots, P_n$ deposit coins and run fair reconstruction (fair exchange) with penalties to swap the $y_i$’s among themselves.
Fair exchange in the $\mathcal{F}^*_{CR}$-hybrid model - the ladder construction

“Abort” attack:

$P_2$ claims without depositing

$P_1 \xrightarrow{w_2, q, \tau} P_2$

$P_2 \xrightarrow{w_1, q, \tau} P_1$

Malicious coalition: $P_1, P_2$ obtain $w_3$ from $P_3$

$P_2$ doesn’t claim the top transaction $P_3$ isn’t compensated
Fair exchange in the $F_{CR}^*$-hybrid model - the ladder construction

“Abort” attack:
$P_2$ claims without deposing

Fair exchange:
$P_1$ claims by revealing $w_1$

$\Rightarrow P_2$ can claim by revealing $w_2$
Fair exchange in the $\mathcal{F}_{CR}^*$-hybrid model - the ladder construction

“Abort” attack:
P2 claims without deposing

Fair exchange:
P1 claims by revealing $w_1$
⇒ P2 can claim by revealing $w_2$

Malicious coalition:
Coalition $P_1, P_2$ obtain $w_3$ from $P_3$
P2 doesn’t claim the top transaction
$P_3$ isn’t compensated
Fair exchange in the $\mathcal{F}^*_{\text{CR}}$-hybrid model - the ladder construction (contd.)

Fair exchange:

Bottom two levels:
$P_1, P_2$ get compensated by $P_3$

Top two levels:
$P_3$ gets her refunds by revealing $w_3$

Full ladder:
In principle, jointly locking coins for fair exchange can work well:

1. $M = \text{"if } P_1, P_2, P_3, P_4 \text{ sign this message with inputs of coins}(3x) \text{ each then their } 3x \text{ coins are locked into 4 outputs of coins}(3x) \text{ each, where each } P_i \text{ can redeem output } T_i \text{ with a witness } w_i \text{ that satisfies } \phi_i, \text{ and after time } \tau \text{ anyone can divide an unredeemed output } T_i \text{ equally to } \{P_1, P_2, P_3, P_4\} \setminus \{P_i\}\"$

2. $P_1, P_2, P_3, P_4 \text{ sign } M \text{ and broadcast it, and after } M \text{ is confirmed, each } P_i \text{ redeems coins}(x) \text{ by revealing } w_i$
Unfortunately, $F^*_{ML}$ cannot be implemented in vanilla Bitcoin because of self-imposed “transaction malleability” (ECDSA is a randomized signature algorithm).

Instead, we propose a protocol enhancement that eliminates transaction malleability while retaining expressibility.
Unfortunately, $\mathcal{F}_{\text{ML}}^*$ cannot be implemented in vanilla Bitcoin because of self-imposed “transaction malleability” (ECDSA is a randomized signature algorithm).

Instead, we propose a protocol enhancement that eliminates transaction malleability while retaining expressibility.

Recap:

- $\mathcal{F}_{\text{ML}}^*$ requires $O(1)$ Bitcoin rounds and $O(n^2)$ transaction data (and $O(n^2)$ signature operations), while the ladder requires $O(n)$ Bitcoin rounds and $O(n)$ transactions.

- Multiparty fair computation can be implemented in Bitcoin via the ladder construction.

- Multiparty fair computation can be implemented via $\mathcal{F}_{\text{ML}}^*$ with an enhanced Bitcoin protocol.
Comparison with other ways to achieve fairness

Gradual release

- Release the output bit by bit...
- Even with only 2 parties, the number of rounds depends on a security parameter.
- Complexity blowup because the protocol must ensure that the parties don’t release junk bits.
- Assumptions on the computational power of the parties, sequential puzzles to avoid parallelization.
Comparison with other ways to achieve fairness

Gradual release

- Release the output bit by bit...
- Even with only 2 parties, the number of rounds depends on a security parameter.
- Complexity blowup because the protocol must ensure that the parties don’t release junk bits.
- Assumptions on the computational power of the parties, sequential puzzles to avoid parallelization.

Trusted bank

- Legally Enforceable Fairness [Lindell 2008]
- Requires a trusted party to provide an ideal bank functionality.
- 2-party only: the bank can provide $\mathcal{F}^*_{\text{CR}}$ or $\mathcal{F}^*_{\text{ML}}$ to use our constructions directly, or implement similar protocols.
- Not a secure cash distribution protocol...
Secure cash distribution and poker

How to Use Bitcoin to Play Decentralized Poker

Iddo Bentov
Technion

Ranjit Kumaresan
MIT

Tal Moran
IDC

CCS 2015
“Paradoxical” Abilities 1983-

- Exchanging Secret Messages without Ever Meeting
- Simultaneous Contract Signing Over the Phone
- Generating exponentially long pseudo random strings indistinguishable from random
- Proving a theorem without revealing the proof
- Playing any digital game without referees
- Private Information Retrieval
Secure cash distribution with penalties

Ideal 2-party secure (non-reactive) cash distribution functionality:

1. Wait to receive \((x_1, \text{coins}(d_1))\) from \(P_1\) and \((x_2, \text{coins}(d_2))\) from \(P_2\).
2. Compute \((y, v) \leftarrow f(x_1, x_2, d_1, d_2)\).
3. Send \((y, \text{coins}(v))\) to \(P_1\) and \((y, \text{coins}(d_1 + d_2 - v))\) to \(P_2\).
Secure cash distribution with penalties

Ideal 2-party secure (non-reactive) cash distribution functionality:

1. Wait to receive \((x_1, \text{coins}(d_1))\) from \(P_1\) and \((x_2, \text{coins}(d_2))\) from \(P_2\).
2. Compute \((y, v) \leftarrow f(x_1, x_2, d_1, d_2)\).
3. Send \((y, \text{coins}(v))\) to \(P_1\) and \((y, \text{coins}(d_1 + d_2 - v))\) to \(P_2\).

- In the general case, each party \(P_i\) has input \((x_i, \text{coins}(d_i))\) and receives output \((y, \text{coins}(v_i))\).
- Use-cases: generalized lottery, incentivized computation, \ldots
Blackbox secure cash distribution

- Blackbox realization of secure cash distribution in the $\mathcal{F}_{\text{CR}}^*$-hybrid model.
- Assume: the input coins amount of $P_i$ is an $m_i$-bit number.

**Step 1: commit to random secrets (preprocessing)**

For all $i \in [n], j \in [n] \setminus \{i\}, k \in [m_i]$:  
- $P_i$ picks a random witness $w_{i,j,k} \leftarrow \{0, 1\}^{\lambda}$
- $P_i$ computes $c_{i,j,k} \leftarrow \text{commit}(1^{\lambda}, w_{i,j,k})$.
- $P_i$ sends $c_{i,j,k}$ to all parties.
- $P_i$ makes an $\mathcal{F}_{\text{CR}}^*$ transaction $P_i \xrightarrow{w_{i,j,k}} 2^k, \tau \xrightarrow{} P_j$
Blackbox secure cash distribution (contd.)

Denote the input coin amounts by \( d = (d_1, \ldots, d_n) \) and the string inputs by \( (x_1, x_2, \ldots, x_n) \).

**Step 2: compute the cash distribution**

Invoke secure SFE (unfair for now) for the cash distribution:

- Compute the output coin amounts \( v = (v_1, v_2, \ldots, v_n) \).
- Derive numbers \( b_{i,j} \) that specify how many coins \( P_i \) needs to send \( P_j \) according to the input coins \( d \) and output coins \( v \).
- Let \( (b_{i,j,1}, b_{i,j,2}, \ldots, b_{i,j,m_i}) \) be the binary expansion of \( b_{i,j} \).
- For all \( i, j, k \), if \( b_{i,j,k} = 1 \) then concatenate to the output a value \( w'_{i,j,k} \) that satisfies \( \text{commit}(1^\lambda, w'_{i,j,k}) = c_{i,j,k} \).
- Compute \( y = f(x_1, x_2, \ldots, x_n) \) and output \( y \) too.

Then, use fair exchange with penalties (with time limit < \( \tau \)) to deliver the output to all parties, so that \( \mathcal{F}^*_{\text{CR}} \) claims will ensue.
Is one-shot protocol enough?

Are we there yet?
Is one-shot protocol enough?

Are we there yet? In the case of poker, not really.

- The most natural formulation of poker is as a reactive secure MPC.
- Multistage protocol: after each stage of the computation some intermediate outputs are revealed to the parties.
  - Example: the top card of the deck is revealed to all parties.
- One-shot protocol is not the natural formulation:
  - A circuit that takes into account all the possible variables is highly inefficient.
  - Those variables may depend on external events (say, you receive a phone call regarding an unrelated financial loss).
  - $\Rightarrow$ must be dropout-tolerant:
    - After a stage that reveals information, corrupt parties must be penalized if they abort.
    - In fact, the corrupt parties must be penalized unless they continue the next stage of the computation.
Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.
Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.
- The given secure MPC (whitebox) where for every round $r$ a single message is broadcast by a designated party $P_{ir}$.
Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.

- The given secure MPC (whitebox) where for every round $r$ a single message is broadcast by a designated party $P_i_r$.

- $\mathcal{F}_{CR}^*$ transactions $P_i \xrightarrow{\phi_{i,j}} P_j$ where $\phi_{i,j}$ is a circuit (script) that is satisfied if $P_i$ created multiple signed extensions of protocol’s execution (with a unique starting nonce).
Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.

- The given secure MPC (whitebox) where for every round \( r \) a single message is broadcast by a designated party \( P_{i_r} \).

\[ F_{CR}^* \text{ transactions } P_i \xrightarrow{\phi_{i,j}} P_j \text{ where } \phi_{i,j} \text{ is a circuit (script) that is satisfied if } P_i \text{ created multiple signed extensions of protocol’s execution (with a unique starting nonce).} \]

- Blackbox secure cash distribution as described, with refunds at time \( \tau \) that exceeds the see-saw time limits, and hence with circuits specified at start that are utilized in the final rounds.
The see-saw construction: 2 parties

**Roof deposit.**

\[
P_1 \xrightarrow[q,\tau_m,2]{TT_{m,2}} P_2 \quad (Tx_{m,2})
\]

**See-saw deposits.** For \( r = m - 1 \) to 1:

\[
P_2 \xrightarrow[2q,\tau_{r+1,1}]{TT_{r+1,1}} P_1 \quad (Tx_{r+1,1})
\]

\[
P_1 \xrightarrow[2q,\tau_{r,2}]{TT_{r,2}} P_2 \quad (Tx_{r,2})
\]

**Floor deposit.**

\[
P_2 \xrightarrow[q,\tau_{1,1}]{TT_{1,1}} P_1 \quad (Tx_{1,1})
\]
The see-saw construction: multiparty

**Roof deposits.** For each $j \in [n-1]$:  

\[
P_j \xrightarrow[\text{TT}_n]{} q, \tau_{2n-2} \rightarrow P_n
\]

**Ladder deposits.** For $i = n-1$ down to 2:

- **Rung unlock:** For $j = n$ down to $i + 1$:

\[
P_j \xrightarrow[\text{TT}_i \land U_{i,j}]{} q, \tau_{2i-1} \rightarrow P_i
\]

- **Rung climb:**

\[
P_{i+1} \xrightarrow[\text{TT}_i]{} i \cdot q, \tau_{2i-2} \rightarrow P_i
\]

- **Rung lock:** For each $j = n$ down to $i + 1$:

\[
P_i \xrightarrow[\text{TT}_{i-1} \land U_{i,j}]{} q, \tau_{2i-2} \rightarrow P_j
\]

**Foot deposit.**

\[
P_2 \xrightarrow[\text{TT}_1]{} q, \tau_1 \rightarrow P_1
\]
The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With $m$ rounds, $O(n^2m)$ calls to $F_{CR}^*$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With $m$ rounds, $O(n^2m)$ calls to $\mathcal{F}_{\text{CR}}^*$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
- Party $P_i$ who aborts pays compensation to all other parties.
- In the ladder $P_i$ can abort and then nobody learns the secret.
The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With \( m \) rounds, \( O(n^2m) \) calls to \( \mathcal{F}_{\text{CR}}^* \) (ladder is \( O(nm) \)).
- \( O(nm) \) security deposit by each party.
- Party \( P_i \) who aborts pays compensation to all other parties.
- In the ladder \( P_i \) can abort and then nobody learns the secret.
- This is crucial for reactive functionalities:
  - Consider poker: suppose that in round \( j \) all parties exchange shares to reveal the top card of the deck.
  - If \( P_i \) didn’t like this top card, we must not allow \( P_i \) to abort in round \( j + 1 \) without punishment.
The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With $m$ rounds, $O(n^2m)$ calls to $\mathcal{F}^*_{\text{CR}}$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
- Party $P_i$ who aborts pays compensation to all other parties.
- In the ladder $P_i$ can abort and then nobody learns the secret.
- This is crucial for reactive functionalities:
  - Consider poker: suppose that in round $j$ all parties exchange shares to reveal the top card of the deck.
  - If $P_i$ didn’t like this top card, we must not allow $P_i$ to abort in round $j + 1$ without punishment.
- The circuits verify a signed extension of the entire execution transcript, and that this extension conforms with the protocol.
- $\Rightarrow$ needs more expressive scripting language than vanilla Bitcoin, but not Turing complete scripts because the round bounds are known in advance.
The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.
The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.
- Preprocessing step: make the cash distribution transactions with random circuits $w_{i,j,k}$.
- Invoke (preprocess) at start an unfair SFE that:
  - Shuffles the deck according to the parties’ random inputs.
  - Computes commitments to shares of all the cards.
  - Deals shares of the hands and shares of the rest of the cards to all parties, and also delivers all the commitments to all parties.
The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.
- Preprocessing step: make the cash distribution transactions with random circuits $w_{i,j,k}$.
- Invoke (preprocess) at start an unfair SFE that:
  - Shuffles the deck according to the parties’ random inputs.
  - Computes commitments to shares of all the cards.
  - Deals shares of the hands and shares of the rest of the cards to all parties, and also delivers all the commitments to all parties.
- The $F_{CR}^*$ circuit in each round of the see-saw will verify signatures of a transcript, then enforce betting rules or force a party to reveal a share of a card, or in the final round force a party to reveal some $w_{i,j,k}$ values.
- For example: if all partied called and the top card on the deck should be revealed, then the next see-saw circuits will require each party to reveal her share of the top card.
Some open questions

- Lower bound of linear number of rounds for fairness with penalties in the $\mathcal{F}_{CR}^*$-hybrid model?
- Constructing secure cash distribution with penalties from blackbox secure MPC and $\mathcal{F}_{CR}^*$?
Some open questions

- Lower bound of linear number of rounds for fairness with penalties in the $\mathcal{F}_{CR}^*$-hybrid model?
- Constructing secure cash distribution with penalties from blackbox secure MPC and $\mathcal{F}_{CR}^*$?

Thank you.